Panels at the 1997 IEEE Symposium on Security and Privacy, Oakland, CA, May 5-7, 1997

by Mary Ellen Zurko, The Open Group Research Institute
(m.zurko@opengroup.org)

This year's symposium contained three panels, each one opening one of the days of the conference. The conference proceedings contain writeups of the panels as well, for anyone interested in information from the horses' mouths. I apologize to those questioners whose names I did not know and therefore cannot associate with their excellent questions.

The first panel, which opened the symposium, was modeled after an Oxford-style debate. The resolution being debated was: The concept of Trusted Computing Base as a basis for constructing systems to meet security requirements is fundamentally flawed and should no longer be used to justify system security architectures.

Arguing in favor were:
Lead: Bob Blakley (IBM)
Second: Darrell Kienzle (U. of Virginia)

Opposed were:
William R. Shockley
Second: LT (USN) James P. Downey (Naval Postgraduate School)

John D. McLean (Naval Research Laboratory) moderated. While John tried to get the audience to vote their inclination with their feet throughout the debate, he was reduced to taking regular polls. The initial vote on the resolution was pretty evenly split three ways between in favor, opposed, and neutral. John pointed out that the definition of TCB being used in this debate is from the Orange Book.

Bob led off, attired in black hat and boots as befit the imagine of the resolution. Bob pointed out that consumers are buying Microsoft Windows 95 (the upgrade being the top selling in the retail business software category in a PC magazine survey), and not a TCB (Windows NT). While surveys indicate that consumers are concerned with security, they're buying anti-virus programs (which were second and third in the same survey). Even buying Windows NT does not produce an evaluated TCB, since most users won't install it in the C2

configuration. In that case, all assurance bets are off.

Bob then discussed the four fatal flaws of TCBs: 1. We can't sell trust
2. We can't build trust
3. Customers can't use trust
4. TCBs can't support a consumer software market

1. "The NSA trusts it so you should too" has not proved to be a compelling evaluation argument to consumers. How do they understand why NSA trusts the system when all they are given is a large amount of highly technical, censored documentation. In addition, consumers read in the media that even trusted systems are broken into. So, Bob's first challenge to the opposing said was: define what it means to trust a system so users can understand it and feel safe with trusted systems.

2. Bob claimed that assurance is impossibly hard and very expensive in a world that is very cost competitive. It slows development. A C2 OS2 didn't look financially promising to IBM.

3. The certification process requires a certifier at the site where the TCB is deployed. The certifier needs to know what it will be used for, but customers aren't good at certification, because they don't know what it will be used for. Trust in the system depends on trust in the administrator. Many systems administered by the end user. The designer, administrator, and certifier must all agree on security objectives, but this is not the case in corporate environments. Internal politics can fragment security goals. Bob's next challenge was to dispute the definition: "a TCB is a set of mechanisms that cause a security exposure when they violate expectations."

4. A TCB needs to define operations and objects for mediation. Each newly installed application adds to these, and so each new application extends the TCB (as the Purple Book extends it to cover databases). This compromises the minimality goal, and forces applications to be designed for a particular TCB and policy.

During the five minutes for follow-up questions, Bob quoted Darrell as saying "If it doesn't work in practice, it doesn't work in theory." Bill stated that if it's not a reference monitor, it can't be effective. Bob suggested the immune system as a counter example.

As an independent consultant, Bill was able to start his portion of the debate off with the statement "the opinions I express are entirely those of my employer." Bill asked if we were talking business or science? His strategy in was to refute the economic and psychological

arguments and propose a counter plea to the application proxy argument, to prove that the TCB can be effectively applied to modern technology.

He stated that the economic argument was scientifically unsound, reducing it to "If you can't sell it, it must be broke." Many systems have and continue to be produced to meet the TCB definitions. No proof was offered that the costs outweigh the benefits. It can't cost millions of dollars to produce a C2 TCB, yet people spend billions on virus checkers and firewalls. The user community is spending more on security than the vendors are spending on TCBs. If you live in a cardboard city, you will spend money on fire extinguishers.

Bill suggested that the cost driver is the level of assurance not the basic (C2) TCB. He was not debating if the NSA is right or the evaluation process is the right one. An alternative paradigm to look to might be an underwriters laboratory underwritten by insurance industry

Bill characterized the affirmative economic argument as:
1. the industry doesn't produce TCBs
2. the industry is a free market
3. the action of a free market is optimal
therefore it is optimal not to produce TCB

However, he claims that the market is oligopolistic. Point 3 implies that you can get optimal allocation of utility. A free market does not optimize wages, provide social security, or spend money on R&D. We have social goals that need to be met nationwide, and you won't meet those with a free market.

To counter the claim that trusted component must induce trust, Bill pointed out that you can produce trust completely in the marketing department. Instead, we should define trust by evaluation by a laboratory.

Bill said there was a need for more inter-domain work, to support architectures such as lots of little single domain PCs for a multi domain system. He concluded by saying that from "everything is inside some TCB" it does not flow that "there is some TCB that everything is inside".

During the 5 minutes of discussion, Bill reiterated that he also does not believe in the monolithic TCB approach; he has been spreading the gospel for application-level policy for many years. Darrell pointed out that even people aware of the risks do not always choose to use a TCB. Bill suggested it was due to the value of the information needing protection, Darrell suggested it was due to trade-offs with other goals.

Since a poll on who had changed their minds was taken at each transition, James led off with the somewhat joking statement, "We have not changed our mind." James discussed their responses to the challenges outline in the proceedings.

In response to the challenge to define trust for normal end users, James asks why should they feel safe? Trust should be technical, so that its merits can be judged. Institutions can be used for protecting the end user (such as airline safety). Independent organizations can audit and accredit secure systems.

In response to the challenge that TCBs cause security exposures, James points out that the TCB must include the administrative input of policy information. There is more user/ administrator interface design work needed, but a TCB must be configurable to be usable

Finally, like Bill, James suggests work on composite, multiple, or extensible TCBs or TCB subsets to support networks and applications.

During the 5 minutes of questioning, Bob suggested that there is no competent authority for making TCBs suitable for their environment given the complex, quickly evolving environment of the Internet or large computer networks. Darrell pointed out that he had never seen two PCs with the same configuration, and James suggested that while end users may configure their home systems for surfing, they might not be allowed to configure their workstations at work.

Darrell started by stating that the TCB contains everything necessary for security, which can ease the verification burden. While this is appealing in theory, is it practically viable? The strengths of the TCB is that it defines a single policy per system, to which all users comply. However, this is overly restrictive for a general model. While no defense is perfect, applications have to trust the TCB. Applications defending themselves put their security outside the TCB.

The old environment of custom built systems, stand alone systems, and a philosophically homogeneous user base masked the TCB's flaws. The TCB's habitat is shrinking. Systems are large and complex, and verification is a virtual impossibility. COTS gives attackers known ways to attack. A single policy may not work when a company is may be forced to cooperate with a competitor. In response to the question of "but what have you got that's better?", Darrell urged research on new paradigms such as those explored in the New Security Paradigms Workshop.

During the 5 minutes of questioning, Bill pointed out that the definition of TCB says it must enforce "a security policy", which is a loophole. TCBs are per-policy.

James suggested that the alternative being put forward for a trusted computing base is a comfy computing base. In response, Darrell warned that it is dangerous to say that users don't know what's best for them.

At this point, the floor was opened up to audience questions and reactions.

Marv Schaefer said that the affirmative team had defeated itself by equating TCB and security with C2, which is easily defeatable by an attacker. Bob replied that they were trying to make the point that even the easy things we sort of know how to do are impractical.

Paul Karger pointed out that to security practitioners around before the OB, that OB was revisionist security. There were only three levels of security: hopeless and don't even bother, limited, really secure. The original theory states that these (C2) systems are hopelessly insecure in the first place. He likened the security industry today to the medical profession in the late 19th century; snake oil and patent medicine. Bob agreed that there were plenty of applications being deployed today that a sensible person would veto.

Terry Benzel rephrased the affirmative's argument as, if can't protect against a free for all, TCBs are no good.

When a questioner suggested that we need the equivalent of safety codes for building boilers, Bob pointed out that boilers are not a general purpose device. You can't anticipate all the things a user might do with a computer.

A questioner asked if can you reproduce or protect everything that might happen. Bill stated that the battle of wits approach can't work, that was established early on. The reference monitor defines the security perimeter with well defined semantics and makes sure each operation is secure; an a priori approach. Bob suggested that this was equivalent to the old joke of a drunk searching for his keys under a lamppost because it was so much brighter there. We define the problem that we know how to fix.

Peter Neumann asked with 60 million lines of code in Star Wars, with a security policy that states bad things don't happen, with injections of electromagnetic interference into smart cards, with runtime libraries, compilers, and networking - where is the TCB? Bill replied that if we build first and look for the TCB second, the TCB will be the whole thing. Having

well modularized software is difficult; we often have code that escapes rather that being released. Bob concluded that the TCB's designer's stance was that the designer is omnipotent, omniscient, and omnipresent, and most of us aren't.

John called for a final vote determined by which door each of us exited by (thereby disallowing any neutrals). The vote as for - 66 and against - 117. (With 256 registered conference attendees, the neutrals may have just stayed in the room.)

The second panel was Ensuring Assurance in Mobile Code.

Moderator: Marv Schaefer (Arca)
Panel Members: Sylvan Pinsky (NSA), Drew Dean (Princeton University), Jim Roskind (Netscape), Li Gong (JavaSoft), Barbara Fox (Microsoft)

Marv introduced the panel with an architectural picture of mobile code executing on a host OS. He asked "What evidence do we have from producers that using these won't leave one vulnerable?" Even the documented paradigm doesn't always work as advertised. What protection claims can be provided to the owners(s) of resources that reside on (or that can be manipulated from) the Java-enabled workstation? How can the user (or resource owner or knowledgable referee) be convinced that it is safe to download and execute applets on his platform? Can claims be made (and supported with credible evidence) about the enforcement of claims made if there exist persistent objects shared among two or more distinct applications or applets? How do you know? How do I know? How long do we think we know? What if the interlopers don't know?

Drew Dean started with The Good, the Bad and the Ugly. He was only looking at the closed system of Java, claiming we have to understand that first.

The Good is there's been a lot of progress in a year. There work on Java type-safety, the JVM bytecode verifier, and dynamic linking.

The Bad (aka not so good) is that we write and reason about Java, but execute JVM bytecode. JVM bytecode strictly richer than Java, and we don't really understand the connection. What are sufficient restrictions on the bytecode to ensure Java semantics? The language and bytecode specifications are incomplete. Surprising questions come up in cleanroom implementations. The bytecode format not designed for verification. It is needlessly hard to specify and needlessly hard to implement.

The Ugly is that penetrate and patch is still the dominant mode of operation. The Java runtimes are huge. JDK 1.1 (for Solaris) is 130,000 lines of C (almost all of this security relevant due to the potential for buffer overflow, etc.) and 380,000 lines of Java. JDK 1.1 has grown a lot compared to JDK 1.0.2. The C portion is approximately 2x as large; the Java portion is approximately 4.5x as large. When a patch comes out, there is all sorts of new code to explore, and of course there's going to be a new bug. Drew concluded by saying that he doesn't know what to do with .5 million lines of code.

Barbara Fox started off saying that the gap we were about to experience is between research and product. She showed a graph of what she called the code safety paradox: while the sandbox is high on the enforcement side, native code is high on functionality. How do we get a good balance? The game is Risk Management not Risk Avoidance. Using the internet/ surfing is a dangerous sport.

She outlined a confidence continuum. It starts with Authenticode, which uses digital signatures to get one point of assurance (the signer, with no revocation). In the middle is Java, which uses digital signatures, an explicit "OK" from the user on what the code does, and a VM. This offers three points of assurance. W3C digital signatures, where detachable labels can be used by anyone to make signed statements about code, offers n points of assurance.

Her goals include:
2. ahead of time checks with detachable labels and filtering, and
3. deterrents such as audit logging, high-assurance certificates (those certified must put some skin in the game), high-visibility revocation, cops and courts.

Li Gong stated that, while Barbara mentioned important sociological aspects, he would focus on the engine and construction. He asked us "how many people does it take to make a high assurance system?" The answer was 1, to make sure that no system is ever built. You have to have a very good, simple, understandable security model for assurance, and map that to an architecture. Then, the implementation must be of high quality. There is no practical way of formally verifying these relationships. You have to sympathize with marketing people; this process doesn't work as a slogan.

Li said that while Drew mentioned that software is getting bigger, its also getting smaller (personal Java, meta Java). It's hard to have a good handle on a quantifiable level of security of system. The proof of correctness will lag behind 5 - 10 years. Li's team is focusing today to make sure that the new architecture helps to reduce the chance of making a mistake. They're trying to help people to work on known problems or issues and to stick

to basic principles. They're taking the past 30 years of published work, distilling the best practices, and applying them. They're hiring good people who know security, and giving out source code for other people to evaluate. They're doing everything they can do as a product team today, and hoping researchers like Marv and Sylvan can help out with long term problems.

I have seen Jim Roskind give essentially the same talk three times now, and two out of three he was "tricked into thinking [it] was a high tech conference" which would support showing slides from a laptop (the two were security conferences; the real high tech conference was the last International WWW conference). Luckily, he did not have to resort to pantomime this time; he had hand written slides. His job is responding to bugs posted in the Wall Street Journal and the New York Times. He's trying to put himself out of a job by making things cleaner and more possibly secure.

He's looking at ergonomics and security. To be secure, the system must be usable. Their target audience is the Internet user with minimal savvy. As with real keys, users don't need to know about internal mechanisms. They don't provide too many dialogs and reuse previous decisions (per the user's request). Dialogs present summary information such as high, medium, or low risk, with the option to see more details. The user must be given fine grain control of grants so they can distinguish between giving out the time of day and power of attorney. They must allow the user to edit and review grants.

They are also looking at programmer ergonomics. They will allow programmers to request the least privilege with a fine grain capability model. They are avoiding the binary trust (all or nothing) model. Their language support encourages holding privilege for the shortest duration by lexically restricting that duration. They can define the security relevant portions of code based on this temporally restricted duration, and automatically remove the privilege for the programmer.

Sylvan Pinsky has been working with Marv on the security context for the Java environment, in a cooperative research program with Javasoft. The security context is a layered approach to software mechanisms: the language and compiler, the bytecode verifier, and the class loader.

They are using a communications analogy: the old fashioned "party line" phone system. Back then we knew the members of the party line. Today, with global connectivity, any user is a potential member of the party line. As Bob Morris said back in '88, to a first approximation, every computer is connected with every other computer. We are becoming massively interconnected. We must coexist with people and systems of unknown and

unidentifiable trustworthiness. As Peter Neumann said, our problems have become international as well as national.

Any meaningful security analysis for mobile code must include the underlying platform OS and the Internet. They are examining security from a system prospective: applets, application, native methods, VM, OS. They would like to provide evidence that a valid JVM cannot be replaced or corrupted though the JVM or underlying platform.

There has been a divergence between the security and Internet communities. The security community focuses on security relevant features encapsulated with a small analyzable mechanism. The Internet provides multi-million lines of code, and the hardware and software boundaries blurred. He is trying to merge these two directions together, using trust technology, security modeling, formal specifications, and formal verification.

At the Java security panel at NISSC, addressing the need to apply security technologies to the 90s, Paul Karger said "this is rocket science!" Marv adds, "with a booster". Paul shouted out "and O rings!".

Sylvan's group is looking at Secure Alpha (a distributed system) and capability-based systems. They want to produce:
1. a high level of security policy
2. identification of environmental assumptions
3. formal specification
4. implementation specification

Marv opened up the floor for questions, beginning with reiterating some of his own from the start. Li mentioned that a new security architecture is due out in early summer. They are trying to do everything in Java. Jim said his strategy is trying to keep his head down lower than the other people's heads. That way, all of our heads will be lower over time. Sylvan pointed out that multiple, diverse systems running across the Internet with diverse security policies pose a big problem in terms of composability of policies.

Marv asked what if sometimes I run one application, and another time run something else? Some set of global invariants might be required. Jim stated that when you allow arbitrary other systems (like sendmail) to adjust file system, you're in a world of hurt. But he questioned, is the consumer interested in ultimate safety? Does it make sense to put a 50lb lock on the front door and still have windows that are very thin? Barbara continued with this theme when she quoted Butler Lampson as saying that he had been been worrying about locks on doors, and should have been working with the cops. Jim suggested a more

distributed analogy: aspirin with tamper resistant covers. Barbara agreed that the idea is to make decisions about things before they happen.

Paul Karger pointed out that the problem of downloadable, executable content is not restricted to Java. There is Postscript, Word documents with viruses, Lotusscript, and Javascript.

Marianne Mueller asked Drew to say specifically, what is hard about verifying the bytecode? Drew said that to a first approximation, a lot of what bytecode verification is doing is type checking. This information could be saved at compile time instead of thrown away.

Bob Blakley asked, if dancing hippos are a 10 and realtime control of nuclear plant is 100; what's the risk we can take with today's technology? Barbara joked that it was 11, then stated that as long as you can give people information for their decisions, they can make an intelligent choice.

Marv pointed out that an intelligently coded attack can detect the environment and who its running on behalf of, and wait until right combination of circumstances to do damage.

One questioner noted that when he locks his front door, he can check the doorknob to verify that its really locked. He can't verify that his access controls are really working. How we give the user confidence? Jim pointed out that ordinary locks are easy to pick, and Barbara emphasized consumer confidence.

John McHugh pointed out that there is a fine line between misfeasance and malfeasance. He's been burned more by the former than the latter. Should we use cops and jails for both? Marv replied, "You'd sell more debuggers." The conversation drifted to the use of audit logs as evidence. Drew pointed out that you just need to attack logs as well, and Marv added that resetting the real time clock is not a privileged operation. Jim agreed that a mistake is more often the problem. He asked, "When your child does bad; do you beat them or set them up with a potential to succeed?" The latter is the intense focus at Netscape.

Li concluded by stating that these are browser companies, and Javasoft is a platform company (which evoked some laughter). Schlumberger has a smart card with 5 meg of memory; it can't pop up a window to solve a problem. The Java security architecture has to work with no file system too (in which case, where is the audit log?).

The last panel was Security in Innovative New Operating Systems, moderated by Cynthia E. Irvine (Naval Postgraduate School). The panel members and their projects were:
Robert Grimm, SPIN Project (University of Washington)
George Necular, FOX Project (Carnegie Mellon University)
David Mazieres, Exokernel Project (MIT)
Oliver Spatscheck, Scout Project (University of Arizona)
Jeff Turner, Fluke Project (University of Utah)

Robert Grimm started by describing SPIN. It provides a minimal core of trusted services and an infrastructure for extensions. It is written in Modula-3. The extensions interact in two ways; they call upon existing services or are called by other services.

A problem is that the extensions form one integral system (they all run in kernel mode). The extensions generally untrusted and used by applications. They need to specify security policies and a way to express and enforce those policies. They settled on Domain and Type Enforcement (DTE) as a flexible MAC system. The access modes are explicit and it allows for controlled changes of privilege. They are working on a formal model of DTE.

The performance benchmark looks good: without security 1.18 sec, with security 1.2 sec. The null call very fast: 0.1 microseconds in the hot cache case. The access check increases the time; a domain transfer increases it even more.

George Necula discussed his "Research on Proof-Carrying Code for Mobil Code Security," with Peter Lee. The problem they are addressing is safety in the presence of untrusted code. Examples of this problem are OS kernel extensions and safe mobile code. They have focused on the intrinsic behavior of untrusted code. They enforce safe access to resources (e. g. memory safety) and resource usage bounds (e.g. bounded termination). They have not focussed on issues like denial of service

The key idea is to shift the burden of safety to the code producer. The code producer must convince the code consumer that the code is safe to run. The producer knows more about the program than the consumer, and it is easier to check the proof than to create it. The consumer defines a formal safety policy and publicizes it. The producer must produce a safety proof and package it with the untrusted application code. The consumer validates the proof with respect to the object code.

The consumer scans code and produces first order predicate logic (like a disassembler). The proof is checked against the transformed code. The proof checking and proof rules work for many kinds of applications.

The benefits of Proof-Carrying Code (PCC) are static checking, onetime cost for safety, and it is more expressive than runtime checking alone. There is early detection of failures (no abort complications), the process is tamper proof, and the consumer code is simple, fast and easy to trust. It can check a wide range of safety and liveness policies. It can be used even for hand-optimized assembly language. You don't have to sacrifice performance for safety.

Their current goals are to test the feasibility and measure the costs of this approach. They chose simple but practical applications written in hand-optimized assembly language. One example was PING. PING's safety policy included statements that the total memory allocated is less than maximum size of the heap, that the buffer to send a packet is a readable buffer of length len, that len <= max packet length, etc.

David Mazieres talk on the Exokernel was called "Secure Applications Need Flexible OSes". They tried to take as much of the code as possible and push it up into untrusted libraries. They end up with raw hardware and a very small Exokernel which tries to interfere as little as possible. They need minimum support for good, discretionary access control. Names for everything are physical (disk blocks, etc.).

They consider the state of multi-user OS security currently hopeless, which is not surprising. Systems must trust way too much software. Trusted software is unreasonably hard to get right. We can make security easier by making it easier to get things right, with better OS interfaces. As an example of unreasonably complex software they give SSH 1.2.12. To fix problems with switching between root and user access, the program was separated into three new programs.

Some of the reasons they give for so many security holes are insecure network protocols, the use of C and libc, violation of least privilege, namespaces decoupled from underlying files (you can be tricked with these), system calls that use all available privileges, no process to process authentication (no way to transfer privileges between processes), and combinations of all of these. To fix these problems, the OS should allow authenticated IPC and credential granting, strip many programs of root privilege (e.g. login), provide access control on low level objects, operate on immutable (e.g. physical) names to avoid races, and a flexible notion of identity.

As an example he discussed their logging process. Each instance of login generates new ID on the fly. Their authentication server runs with a 0 length capability which is effectively root. It generates capabilities for the new ID, granting them back to the login process. Login was never maximally privileged, and a single authentication server replaces many privileged programs.

Oliver Spatscheck presented "Escort: Securing Scout Paths." Scout was designed for running network appliances (firewalls, web servers, etc.). Security policy is determined at the time you build the system, before you compile.

Security in networks is based on explicit information flows restricted by a filter (firewalls). Policy is enforced by endhosts and filters. Security in conventional OSes models each user separately. Each user trusts all the servers. Scout extends the network connection metaphor into the OS. It's called a path. Paths are first class objects. Escort separates paths (information flows) into protection domains. A security filter limits the flow of information on a path if it is traversing multiple protection domains.

Escort's goals are to protect information flow (paths), to support the principle of least privilege, to support most security policies, to support central policy management, to keep globally trusted code small, and to be efficient. Escort is currently being implemented on the DEC Alpha in a single address space with multiple protection domains. The protection domains are enforced by hardware. It has fast user-to-user domain crossing. Its policy is represented as a decision tree, which can be used to represent arbitrary policies.

Jeff Turner presented "Flask: A Secure Fluke Kernel." He is on a one year visit from the NSA to the University of Utah. Fluke is a microkernel designed to take Mach even further. It supports fine-grained decomposition of services, customizable OS functionality, a recursive VM model, hierarchical resource control, state encapsulation, and flexible composition. Every application has a TCB, down to realtime applications on the kernel. Building on the work of Synergy/DTOS, they focus on flexibility and customizability by separating policy from enforcement. They decompose the OS as much as possible. They began with Mach 3.0 as the foundation in 1992.

Flask goals focus on flexible, customizable security. They continue to rely on hardware protection. They support fine-granularity access control and least privilege, and they provide a system-wide solution for downloaded code (such as Java, Activex, Inferno, Juice, Tcl, and plug-ins).

This work required changes to Fluke. All servers need to bind security information to their basic objects, use the new interfaces to create objects given a SID, and interact with the security policy. Microkernel specific changes include adding controls over object transfer, attaching the sender SID to all messages, and allowing the specification of a receiver SID. All servers are policy neutral or policy flexible.

They are done with the microkernel and security server. They have designed secure file and network services and the secure network service. The process manager, VMM, and Flask specification are in progress. They're planning on a code escape in late July of this year.

The research issues of interest include determining what yardstick to use for assurance, good performance, a trust model and policy extensions in sandboxes to support secure containers for downloaded code, determining the interaction of differing policies, and support for new "WebOS" ideas (e.g. rent-a-computer).

At this point the floor was opened up to questions and comments.

Paul Karger asked how the PCC system knows that the predicate they're trying to prove means its secure. George replied that the code consumer determines what's secure with the published safety policy. Paul also noted that the Exokernel has a flavor of the early days of Multics. David agreed that most of the ideas are common sense.

Bill Shockley pointed out that the idea in FOX that you can proof-lock something as a general notion of spraypainting or cryptographic locking is an important theoretical idea that deserves more attention.

One questioner stated that notions such as hardware enforced protection domains are very conventional security models. He asked, is that the right for network appliances? Can you expect the hardware to provide protection domains? Oliver pointed out that the notion of information flow is what Scout protects. If there is no stable storage on a web TV, that is the only thing worth protecting. They use hardware-based protection domains for a higher level of assurance for less effort.

Maureen Stillman asked what theorem prover and what languages does FOX support? FOX uses its own theorem prover and only supports DEC Alpha assembly language.

Cynthia Irvine asked, "Where is the TCB? Can you identify the security perimeter statically?" FOX's TCB is the proof checker, proof rules, VCgenerator that transforms the code. In SPIN, the people who configure it define the boundary. The Exokernel itself is probably the TCB. In Scout, you have to trust the code that does the separation of policy enforcement of paths. Each information flow defines its own TCB. Fluke has a different TCB per application or per situation.

John McHugh asked, "How does the end user know what the policy is? How does the user gain assurance of proper enforcement?". In Fluke, when you add a new piece, you add a

new set of access control permissions. For any application, you specify the policy and plug it in. In Scout, how policy is specified is not part of the kernel; there are no fixed rules there. It's defined by the person who builds the system. In the Exokernel, the application can do whatever it wants with its data or state. In SPIN, the standard TCB is provided and includes assurance that this is secure. You can do something different if you want to. In FOX, each extension comes with a safety policy. Jeff pointed out that another class of decisions are based on history or time, and so need either dynamic checking or the ability to revoke and reverify.

One questioner suggested that these systems shift burden of developing complex security mechanisms to the programmer who is likely to mess it up. Scout plans to ship tools and guidelines that make sense for most of population. David pointed out that you can provide the same interfaces with the Exokernel with a library. Sometimes what the kernel offers is inadequate for what the application needs.

Paul Karger proposed the scenario where an end user goes to the store and buys an Exokernel and bunches of applications. The end user can barely spell policy; why do Acme's stuff and Ajax's stuff running together provide a consistent security policy? How can I compose them and expect them to do anything? Robert asked how you can expect anything consistent when you do the same thing on a Mac system, and Paul said you don't.

Virgil Gligor asked if FOX supported mutual suspicion. Can the producer have requirements on the consumer? It does not.

Carl Landwehr asked what authentication each of these systems use, and whether they could authenticate each other. PCC avoids authentication. SPIN has but their effort into access control, not on authentication. The Exokernel does password authentication because its simple to do. Scout authenticates the default path. Flask allows login with a password or across the net. Carl went on to ask How much trust is vested in the login process? Jeff replied, I'd say lots.

Peter Neumann suggested that they were redefining trust technology to trussed technology: belt, suspenders, smoke and mirrors. He then asked, with his Risks hat on, How would you subvert your system? George would try to find a bug in the implementation of his FOX proof checker. He pointed out that the architecture is proven, and that the checker is 5 pages of C code. Oliver would try to subvert Scout's boot loader or firmware. Jeff would confuse people about Flask's policy. David would attempt trojan horse attacks through NFS on the Exokernel's development. Robert would try to find a way to change SPIN's DTE access matrix.